Chapter 2
A Recovery-Oriented Approach for Software Fault Diagnosis in Complex Critical Systems

Gabriella Carrozza
SESM s.c.a.r.l. - a Finmeccanica Company, Italy

Roberto Natella
Università degli Studi di Napoli Federico II, Italy

ABSTRACT

This paper proposes an approach to software faults diagnosis in complex fault tolerant systems, encompassing the phases of error detection, fault location, and system recovery. Errors are detected in the first phase, exploiting the operating system support. Faults are identified during the location phase, through a machine learning based approach. Then, the best recovery action is triggered once the fault is located. Feedback actions are also used during the location phase to improve detection quality over time. A real world application from the Air Traffic Control field has been used as case study for evaluating the proposed approach. Experimental results, achieved by means of fault injection, show that the diagnosis engine is able to diagnose faults with high accuracy and at a low overhead.

INTRODUCTION

Hardware and software technologies are progressing fast, increasing the complexity of modern computer systems significantly. Even in the context of critical scenarios, we are witnessing a paradigm shift from stand-alone and centralized systems toward large-scale and distributed infrastructures and simple monolithic programs are letting the field to modular software architectures, typically based on Off-The-Shelf (OTS) software items. This allows industries to increase market competitiveness by lowering development costs and reducing the time to market. Testing and veri-

DOI: 10.4018/978-1-4666-2056-8.ch002

fication, along with fault tolerance techniques, are used to satisfy dependability requirements. The key for achieving fault tolerance is the ability to accurately detect, diagnose, and recover from faults during system operation.

The great research effort striven in fault tolerant systems has provided good results with respect to hardware-related errors. Recent examples are (Serafini, Bondavalli, & Suri, 2007), (Bondavalli, Chiaradonna, Cotroneo, & Romano, 2004). However, it is well known that many systems outages are due to software faults (Gray, 1985), i.e., to bugs lying into the code which have, then, a permanent in nature. This means that, if a program contains a bug, any circumstances that cause it to fail once will always cause it to fail, and this is the reason why software failures are often referred to as "systematic failures" (Littlewood & Strigini, 2000). However, the *failure process*, i.e., the way the bugs are activated, is not deterministic since (i) the sequence of inputs cannot be predicted, hence it is not possible to establish which are the program's faults (and failures), and (ii) software failures can be due to environmental conditions (e.g., timing and load profile) which let a given fault to be activated. For this reason, it is said that software faults can manifest transiently. By failure we intend the software modules/components failure in which the fault has been activated. This can be viewed as fault from the whole system point of view (Joshi, Hiltunen, Sanders, & Schlichting, 2005). Activating conditions which cause a software fault to surface into a failure have been recognized to be crucial in (Chillarege et al., 1992), where they are defined as "triggers" and where software bugs are grouped into orthogonal, non overlapping, defect types (Orthogonal Defect Classification, ODC). Software faults which manifest permanently, also known as *Bohrbugs*, are likely to fix and discover during the pre-operational phases of system life cycle (e.g., structured design, design review, quality assurance, unit, component and integration testing, alpha/beta test), as well as by means of traditional debugging techniques. Conversely, software faults which manifest transiently, also known as *Heisenbugs*, cannot be reproduced systematically (Huang, Jalote, & Kintala, 1994), and they have been demonstrated to be the major cause of failures in software systems, especially during the system operational phase (Sullivan & Chillarege, 1991; Chillarege, Biyani, & Rosenthal,1995; Xu, Kalbarczyc, & Iyer, 1999).

Focus in this work is on recovery oriented software fault diagnosis in complex fault tolerant systems. Little attention has been paid so far to this problem, which plays a key role in maintaining system health and in preserving fault tolerance capabilities. Previous studies on software diagnosis aimed to identify software defects from their manifestations through off-line and/or on-site analysis (Tucek, Lu, Huang, Xanthos, & Zhou, 2007). They aim to discover bugs in the code, by using static/dynamic code screening, in order to perform more effective maintenance operations. Thus, they are not able to catch Heisenbugs since this way the underlying environmental conditions are not easy to localize into the code.

In this work, the aim of diagnosis is twofold. First, starting from outward symptoms we are interested in identifying what are the execution misbehaviors which caused failure occurrence, and where these misbehaviors come from, in order to trigger proper recovery actions. This is crucial in complex, modular and distributed systems, for which the overall failure can be avoided by confining and masking the failures of the parts (nodes, components, processes). Second, we aim to provide information about manifested symptoms that are useful for off-line maintenance activities.

The massive presence of OTS items, whose well-known dependability pitfalls do not hold industries back from their usage in critical systems, further exacerbates the diagnosis problem. In fact, faults can propagate in several ways and among several components, depending on a complex combination of their internal state and of the execution environment. Actually, the failures resulting from unexpected faults, known as production